



# DW\_fifo\_s1\_sf

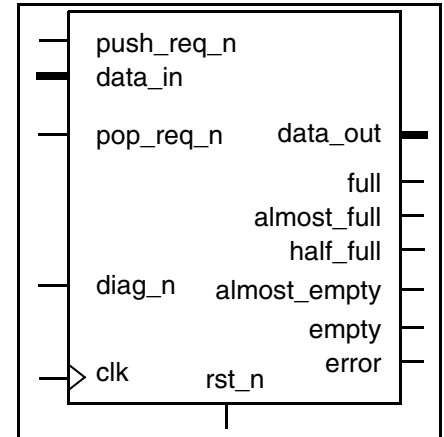
## Synchronous (Single Clock) FIFO with Static Flags

Version, STAR and Download Information: [IP Directory](#)

### Features and Benefits

- Fully registered synchronous flag output ports
- D flip-flop-based memory array for high testability
- All operations execute in a single clock cycle
- FIFO empty, half full, and full flags
- FIFO error flag indicating underflow, overflow, and pointer corruption
- Parameterized word width
- Parameterized word depth
- Parameterized almost full and almost empty flags
- Parameterized reset mode (synchronous or asynchronous, memory array initialized or not)

### Revision History



### Description

DW\_fifo\_s1\_sf is a fully synchronous, single-clocked FIFO. It combines the DW\_fifoctrl\_s1\_df FIFO controller and the DW\_ram\_r\_w\_s\_dff flip-flop-based RAM DesignWare components.

The FIFO provides parameterized width and depth, and a full complement of flags: full, almost full, half full, almost empty, empty, and error.

Reset can be selected at instantiation to be either synchronous or asynchronous, and can either include or exclude the RAM array.

The DW\_fifo\_s1\_sf is recommended for relatively small configurations. For large FIFOs, you should consider using the DW\_fifoctrl\_s1\_sf in conjunction with a compiled, full-custom RAM array.

**Table 1-1 Pin Description**

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset input, active low asynchronous if <i>rst_mode</i> = 0 or 2, synchronous if <i>rst_mode</i> = 1 or 3
push_req_n	1 bit	Input	FIFO push request, active low
pop_req_n	1 bit	Input	FIFO pop request, active low
diag_n	1 bit	Input	Diagnostic control, active low

**Table 1-1 Pin Description (Continued)**

Pin Name	Width	Direction	Function
data_in	<i>width</i> bits	Input	FIFO data to push
empty	1 bit	Output	FIFO empty output, active high
almost_empty	1 bit	Output	FIFO almost empty output, active high
half_full	1 bit	Output	FIFO half full output, active high
almost_full	1 bit	Output	FIFO almost full output, active high
full	1 bit	Output	FIFO full output, active high
error	1 bit	Output	FIFO error output, active high
data_out	<i>width</i> bits	Output	FIFO data to pop

**Table 1-2 Parameter Description**

Parameter	Values	Function
width	1 to 2048 Default: 8	Width of the <code>data_in</code> and <code>data_out</code> buses
depth	2 to 1024 Default: 4	Number of memory elements used in FIFO ( $\text{addr\_width} = \text{ceil}(\log_2(\text{depth}))$ )
ae_level	1 to $\text{depth} - 1$ Default: 1	Almost empty level (the number of words in the FIFO at or below which the <code>almost_empty</code> flag is active)
af_level	1 to $\text{depth} - 1$ Default: 1	Almost full level (the number of empty memory locations in the FIFO at which the <code>almost_full</code> flag is active. Refer to Figure 1.)
err_mode	0 to 2 Default: 0	Error mode <ul style="list-style-type: none"> <li>0: Underflow/overflow and pointer latched checking</li> <li>1: Underflow/overflow latched checking</li> <li>2: Underflow/overflow unlatched checking</li> </ul>
rst_mode	0 to 3 Default: 0	Reset mode <ul style="list-style-type: none"> <li>0: Asynchronous reset including memory</li> <li>1: Synchronous reset including memory</li> <li>2: Asynchronous reset excluding memory</li> <li>3: Synchronous reset excluding memory</li> </ul>

**Table 1-3 Synthesis Implementations**

Implementation Name	Function	License Feature Required
rtl	Synthesis Model	DesignWare

**Table 1-4 Simulation Models**

Model	Function
DW06.DW_FIFO_S1_SF_CFG_SIM	Design unit name for VHDL simulation
dw/dw06/src/DW_fifo_s1_sf_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_fifo_s1_sf.v	Verilog simulation model source code

**Table 1-5 Error Mode Description**

error_mode	Error Types Detected	Error Output	diag_n
0	Underflow/Overflow and Pointer Corruption	Latched	Connected
1	Underflow/Overflow	Latched	N/C
2	Underflow/Overflow	Not Latched	N/C

## Writing to the FIFO (Push)

A push is executed when the `push_req_n` input is asserted (low) and either:

- The `full` flag is inactive (low),

or:

- The `full` flag is active (high), and
- The `pop_req_n` input is asserted (low).

Thus, a push can occur even if the FIFO is full, as long as a pop is executed in the same cycle.

Asserting `push_req_n` in either of the above cases causes the data at the `data_in` port to be written to the next available location in the FIFO. This write occurs on the `clk` following the assertion of `push_req_n`. The data at the `data_in` port must be stable for a setup time before the rising edge of `clk`.

An error occurs if a push is attempted while the FIFO is full. That is, if:

- The `push_req_n` input is asserted (low),
- The `full` flag is active (high), and
- The `pop_req_n` input is inactive (high).

## Reading from the FIFO (Pop)

A pop operation occurs when `pop_req_n` is asserted (low), as long as the FIFO is not empty. Asserting `pop_req_n` causes the internal read pointer to be incremented on the next rising edge of `clk`. Thus, the RAM read data must be captured on the `clk` following the assertion of `pop_req_n`.

Refer to the timing diagrams for details of the pop operation.

An error occurs if:

- The `pop_req_n` input is active (low), and
- The `empty` flag is active (high).

## Simultaneous Push and Pop

Push and pop can occur at the same time if there is data in the FIFO, even when the FIFO is full. With the FIFO not empty, the internal read pointer points to the next address to be popped and the pop data is available at the `data_out` output. When `pop_req_n` and `push_req_n` are both asserted, the following events occur on the next rising edge of `clk`:

- Pop data is captured by the next stage of logic after the FIFO, and
- The new data is pushed into the same location from which the data was popped.

Thus, there is no conflict in a simultaneous push and pop when the FIFO is full. A simultaneous push and pop cannot occur when the FIFO is empty, since there is no pop data to prefetch. However, push data is captured in the FIFO.

## Reset

The `rst_mode` parameter selects whether reset is asynchronous (`rst_mode = 0` or `2`) or synchronous (`rst_mode = 1` or `3`). If an asynchronous mode is selected, asserting `rst_n` (setting it low) immediately causes the internal address pointers to be set to 0, and the flags and error outputs to be initialized. If a synchronous mode is selected, the address pointers, flags, and error outputs are initialized at the rising edge of `clk` after the assertion of `rst_n`.

The error outputs and flags are initialized as follows:

- The `empty` and `almost_empty` are initialized to 1, and
- All other flags and the `error` output are initialized to 0.

If `rst_mode = 0` or `1`, the RAM array is also initialized when `rst_n` is asserted. If `rst_mode = 2` or `3`, only the address pointers, and error and flag outputs are initialized; the RAM array is not initialized.

## Errors

The `err_mode` parameter determines which possible fault conditions are detected, and whether the `error` output remains active until reset or for only the clock cycle in which the error is detected.

When the `err_mode` parameter is set to 0 at design time, the `diag_n` input provides an unconditional synchronous reset to the value of the read pointer. This can be used to intentionally cause the FIFO address pointers to become corrupted, forcing a pointer inconsistency-type error.

For normal operation when *err\_mode* = 0, *diag\_n* should be driven inactive (high). When the *err\_mode* parameter is set to 1 or 2, the *diag\_n* input is ignored (unconnected).

**error**

The *error* output indicates a fault in the operation of the FIFO control logic. There are several possible causes for the *error* output to be activated:

1. Overflow (push and no pop while full).
2. Underflow (pop while empty).
3. Empty pointer mismatch (read pointer  $\neq$  write pointer when empty).
4. Full pointer mismatch (read pointer  $\neq$  write pointer when full).
5. In between pointer mismatch (read pointer = write pointer when neither empty nor full).

When *err\_mode* = 0, all five causes are detected, and the *error* output (once activated) remains active until reset. When *err\_mode* = 1, only causes 1 and 2 are detected, and the *error* output (once activated) remains active until reset. When *err\_mode* = 2, only causes 1 and 2 are detected, and the *error* output only stays active for the clock cycle in which the *error* is detected. For *err\_mode* descriptions, see [Table 1-5](#) on page 3. The *error* output is set low when *rst\_n* is applied.

## Controller Status Flag Outputs

Refer to [Figure 1-1](#) on page 6 for operation of the status flags.

**empty**

The *empty* output indicates that there are no words in the FIFO available to be popped. The *empty* output is set high when *rst\_n* is applied.

**almost\_empty**

The *almost\_empty* output is asserted when there are no more than *ae\_level* words currently in the FIFO available to be popped. The *ae\_level* parameter defines the almost empty threshold. The *almost\_empty* output is useful for preventing the FIFO from underflowing. The *almost\_empty* output is set high when *rst\_n* is applied.

**half\_full**

The *half\_full* output is active (high) when at least half the FIFO memory locations are occupied. The *half\_full* output is set low when *rst\_n* is applied.

**almost\_full**

The *almost\_full* output is asserted when there are no more than *af\_level* empty locations in the FIFO. The *af\_level* parameter defines the almost full threshold and is useful for preventing the FIFO from overflowing. The *almost\_full* output is set low when *rst\_n* is applied.

**full**

The *full* output indicates that the FIFO is full and there is no space available for push data. The *full* output is set low when *rst\_n* is applied.

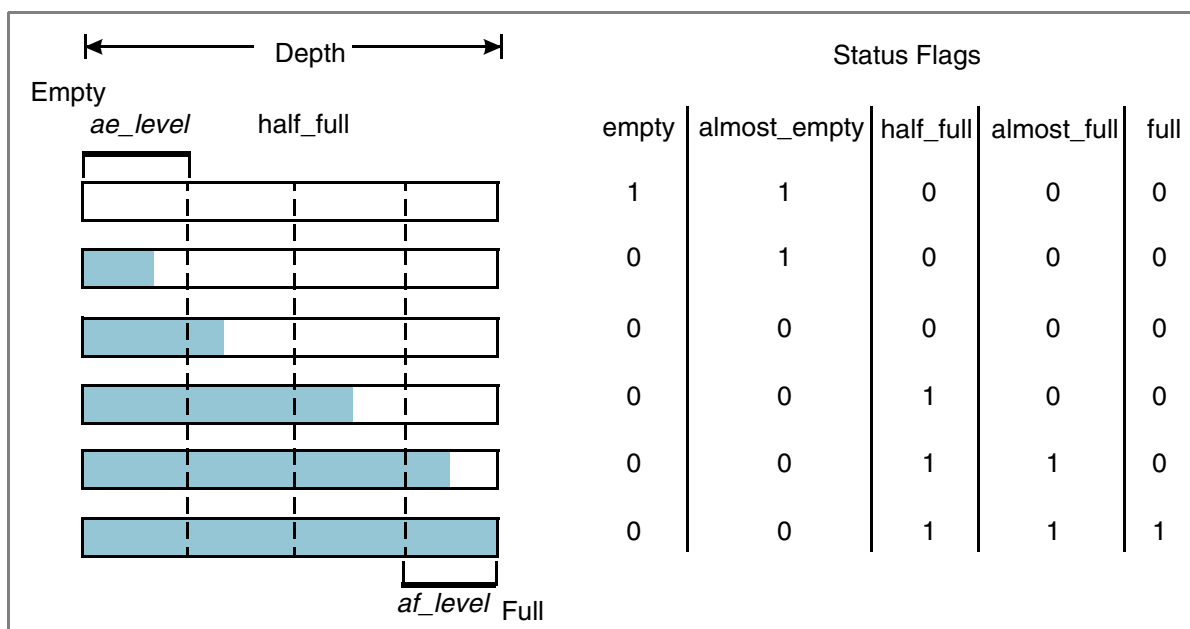
## Application Notes

The *ae\_level* parameter value is chosen at design time to give the input flow control logic enough time to begin pushing data into the FIFO before the last word is popped by the output flow control logic.

The *af\_level* parameter value is chosen at design time to give the output flow control logic enough time to begin popping data out of the FIFO before the FIFO is full. In other situations, this time is needed to cause the input flow control logic to interrupt the pushing of data into the FIFO.

Figure 1-1 shows the status flags of the DW\_fifo\_s1\_sf FIFO at various FIFO storage levels.

**Figure 1-1 DW\_fifo\_s1\_sf FIFO Status Flags**



## Suppressing Warning Messages During Verilog Simulation

The Verilog simulation model includes macros that allow you to suppress warning messages during simulation.

To suppress all warning messages for all DWBB components, define the DW\_SUPPRESS\_WARN macro in either of the following ways:

- Specify the Verilog preprocessing macro in Verilog code:

```
`define DW_SUPPRESS_WARN
```

- Or, include a command line option to the simulator, such as:

```
+define+DW_SUPPRESS_WARN (which is used for the Synopsys VCS simulator)
```

The warning messages for this model include the following:

- If values other than 1 or 0 are present on a clock port, the following message is displayed:

WARNING: <instance\_path>.<clock\_name>\_monitor:  
at time = <timestamp>, Detected unknown value, x, on <clock\_name> input.

To suppress only this warning message for all DWBB components, use the following macro:

- Define the DW\_DISABLE\_CLK\_MONITOR macro. You can define this macro in the following ways:
  - Specify the Verilog preprocessing macro in Verilog code:  
`define DW\_DISABLE\_CLK\_MONITOR
  - Or, include a command line option to the simulator, such as:  
+define+DW\_DISABLE\_CLK\_MONITOR (which is used for the Synopsys VCS simulator)

This message is also suppressed using the DW\_SUPPRESS\_WARN macro explained earlier.

## Timing Waveforms

The following figures show timing diagrams for various conditions of DW\_fifo\_s1\_sf.

**Figure 1-2 Status Flag Timing Waveforms While Pushing**

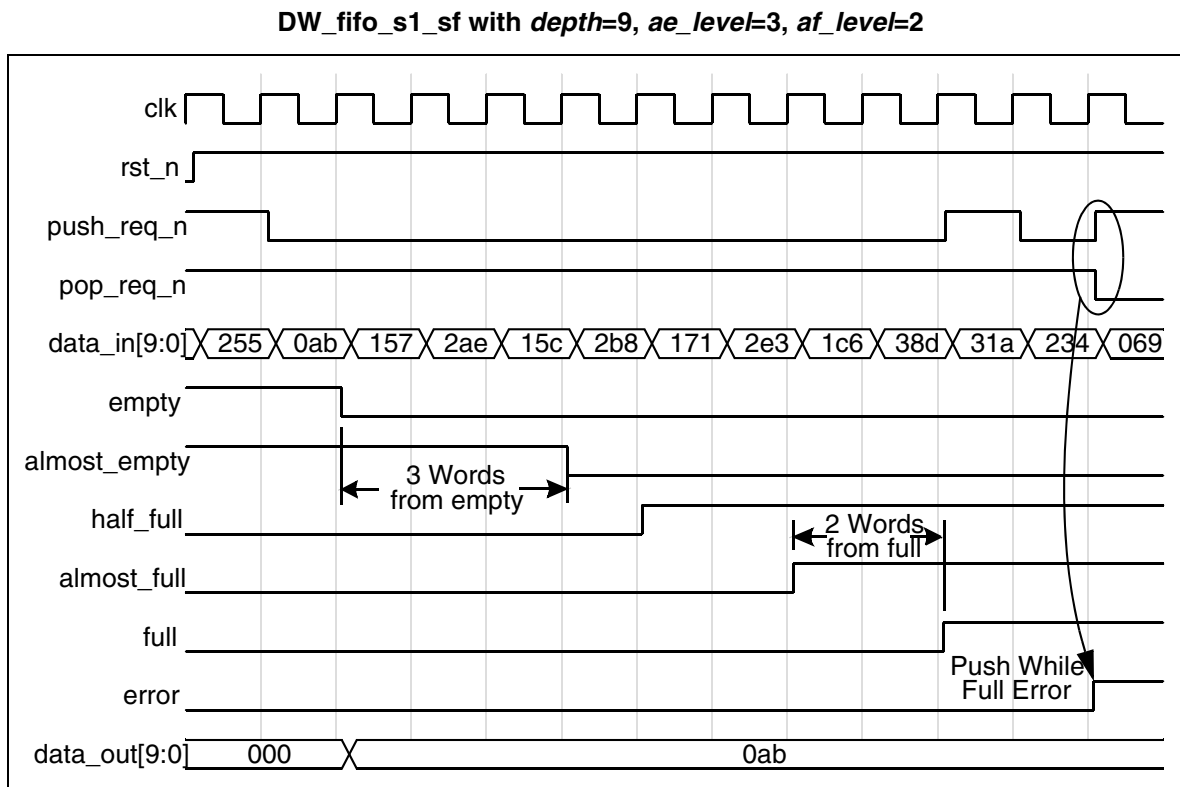


Figure 1-3 Status Flag Timing Waveforms While Popping

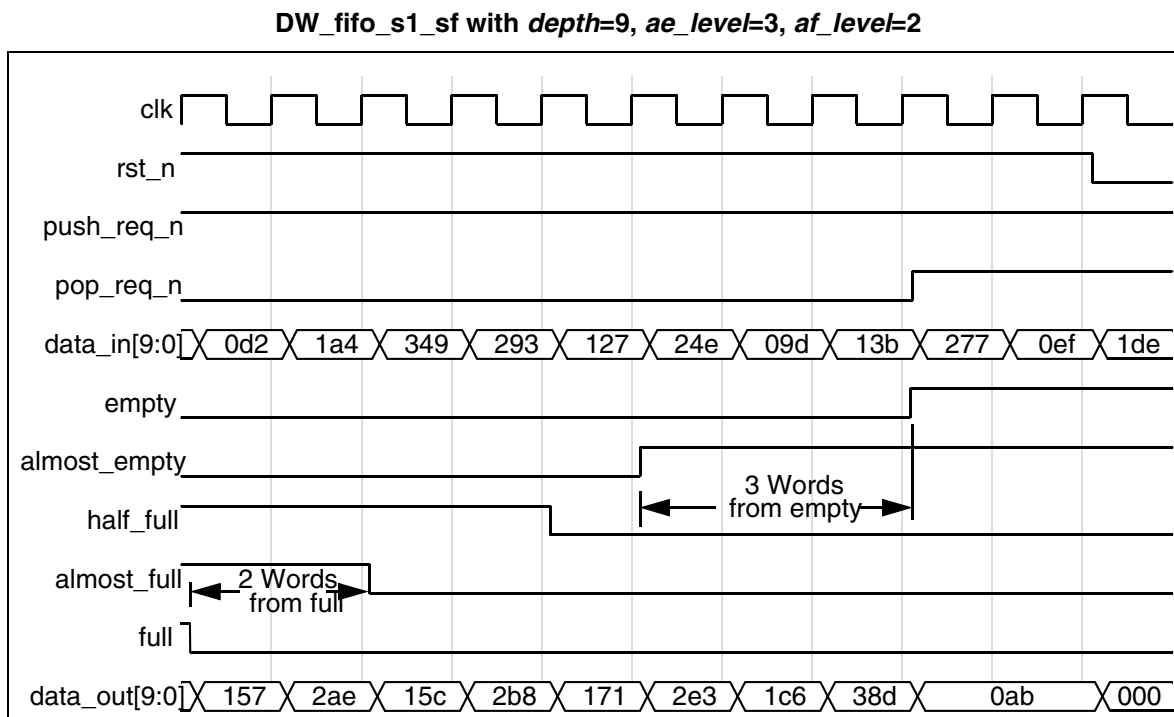




Figure 1-4 Error Flag Timing Waveforms

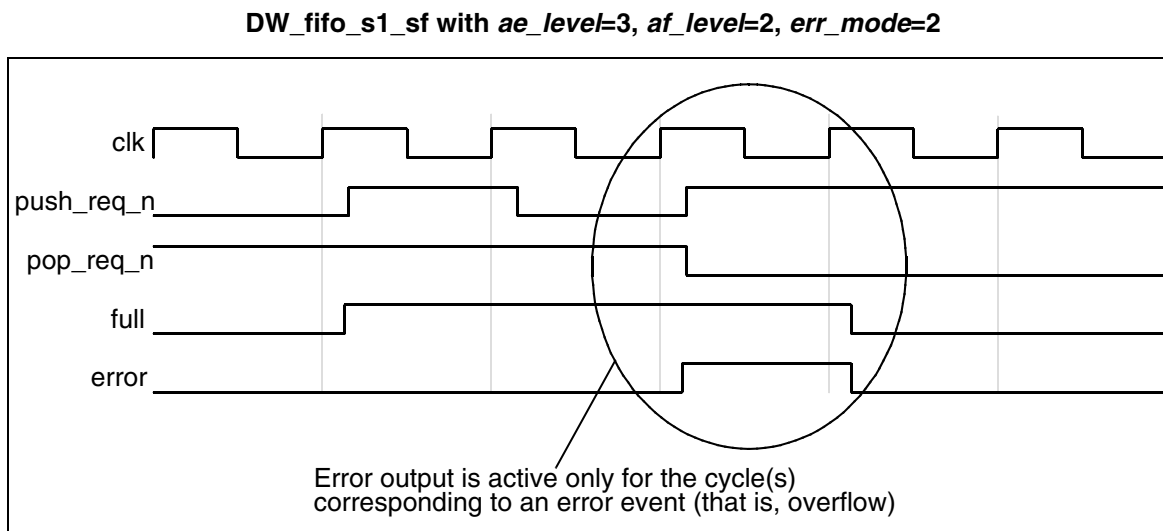
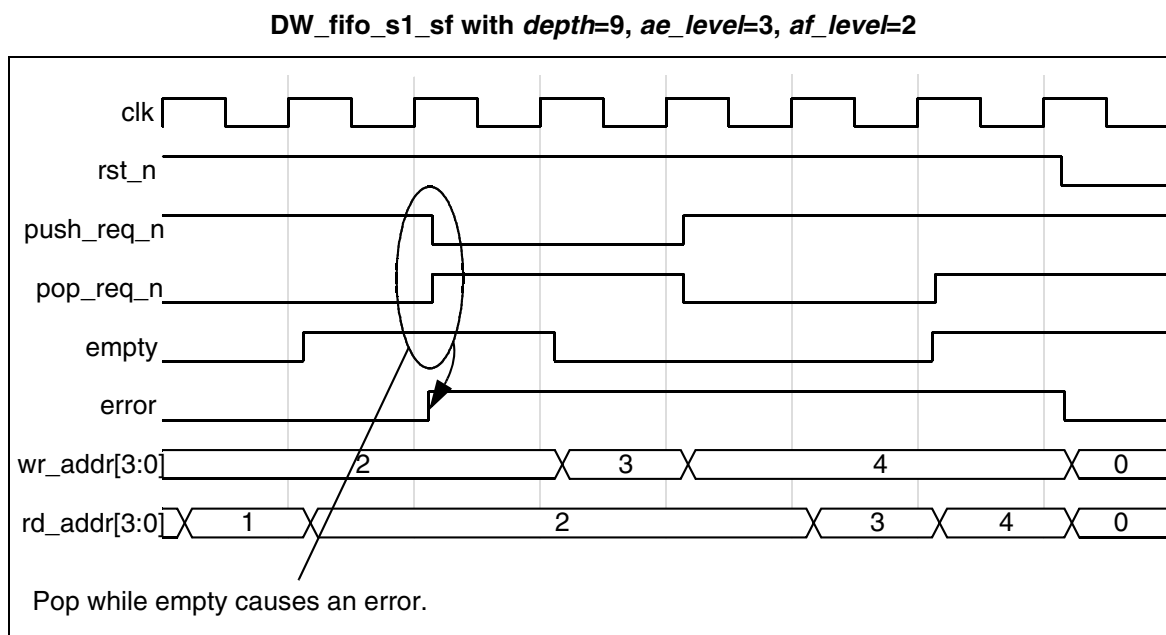


Figure 1-5 Error Flag Timing Waveforms (continued)

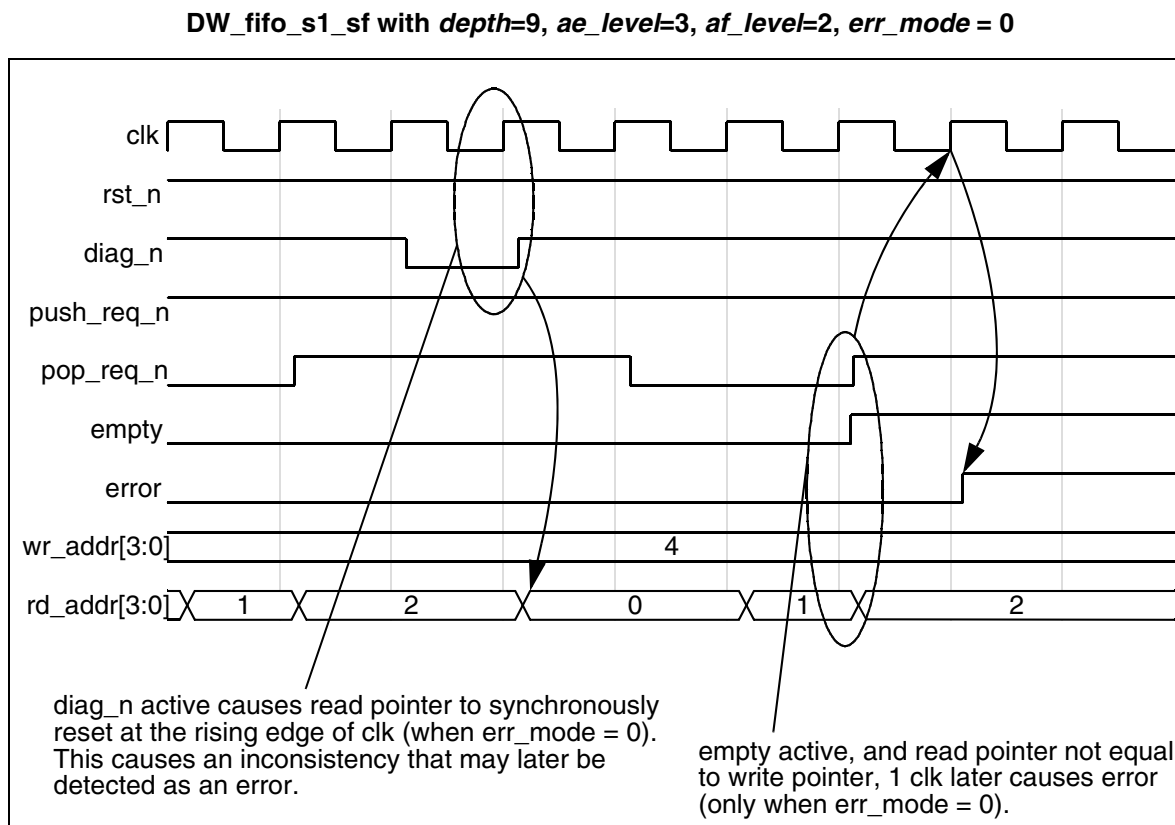


Figure 1-6 Error Flag Timing Waveforms (continued)

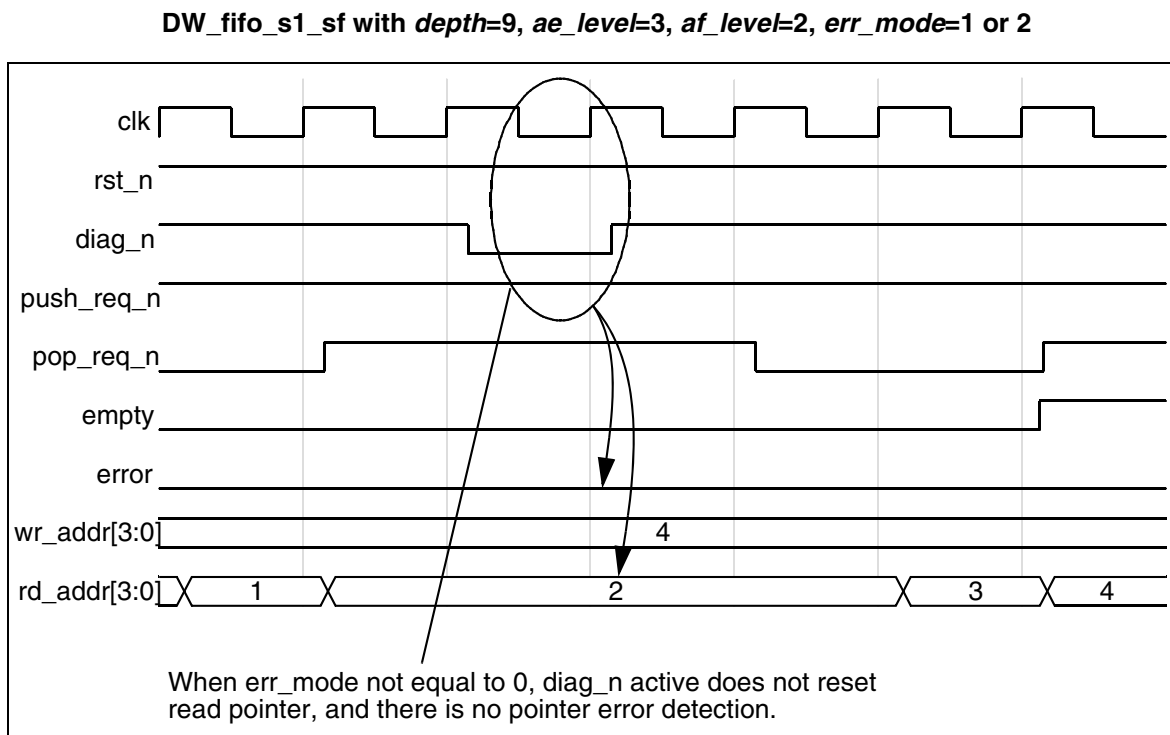
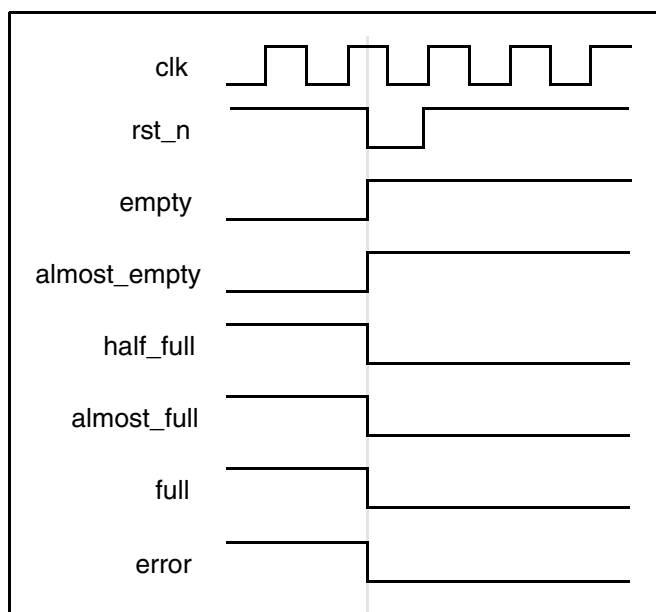
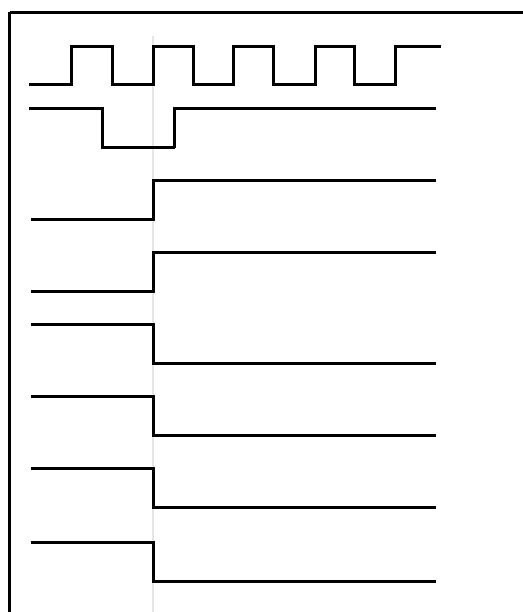


Figure 1-7 Reset Timing Waveforms

**DW\_fifo\_s1\_sf with *depth* = 9, *rst\_mode* = 0 or 2  
(Asynchronous Reset)**



**DW\_fifo\_s1\_sf with *depth* = 9, *rst\_mode* = 1 or 3  
(Synchronous Reset)**



---

## Related Topics

- [Memory - FIFO Overview](#)
- [DesignWare Building Block IP User Guide](#)

## HDL Usage Through Component Instantiation - VHDL

```

library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_foundation_comp.all;

entity DW_fifo_s1_sf_inst is
  generic (inst_width      : INTEGER := 8;
           inst_depth      : INTEGER := 4;
           inst_ae_level   : INTEGER := 1;
           inst_af_level   : INTEGER := 1;
           inst_err_mode   : INTEGER := 0;
           inst_rst_mode   : INTEGER := 0 );
  port (inst_clk           : in std_logic;
        inst_rst_n         : in std_logic;
        inst_push_req_n    : in std_logic;
        inst_pop_req_n     : in std_logic;
        inst_diag_n        : in std_logic;
        inst_data_in        : in std_logic_vector(inst_width-1 downto 0);
        empty_inst         : out std_logic;
        almost_empty_inst  : out std_logic;
        half_full_inst     : out std_logic;
        almost_full_inst   : out std_logic;
        full_inst          : out std_logic;
        error_inst         : out std_logic;
        data_out_inst      : out std_logic_vector(inst_width-1 downto 0) );
end DW_fifo_s1_sf_inst;

architecture inst of DW_fifo_s1_sf_inst is
begin

  -- Instance of DW_fifo_s1_sf
  U1 : DW_fifo_s1_sf
    generic map ( width => inst_width,    depth => inst_depth,
                  ae_level => inst_ae_level,  af_level => inst_af_level,
                  err_mode => inst_err_mode,  rst_mode => inst_rst_mode )
    port map ( clk => inst_clk,    rst_n => inst_rst_n,
               push_req_n => inst_push_req_n,  pop_req_n => inst_pop_req_n,
               diag_n => inst_diag_n,  data_in => inst_data_in,
               empty => empty_inst,  almost_empty => almost_empty_inst,
               half_full => half_full_inst,  almost_full => almost_full_inst,
               full => full_inst,  error => error_inst,
               data_out => data_out_inst );

end inst;

-- pragma translate_off
configuration DW_fifo_s1_sf_inst_cfg_inst of DW_fifo_s1_sf_inst is
  for inst

```

---

```
    end for; -- inst
end DW_fifo_s1_sf_inst_cfg_inst;
-- pragma translate_on
```

## HDL Usage Through Component Instantiation - Verilog

```

module DW_fifo_s1_sf_inst(inst_clk, inst_rst_n, inst_push_req_n,
                          inst_pop_req_n, inst_diag_n, inst_data_in,
                          empty_inst, almost_empty_inst, half_full_inst,
                          almost_full_inst, full_inst, error_inst,
                          data_out_inst );

    parameter width = 8;
    parameter depth = 4;
    parameter ae_level = 1;
    parameter af_level = 1;
    parameter err_mode = 0;
    parameter rst_mode = 0;

    input inst_clk;
    input inst_rst_n;
    input inst_push_req_n;
    input inst_pop_req_n;
    input inst_diag_n;
    input [width-1 : 0] inst_data_in;
    output empty_inst;
    output almost_empty_inst;
    output half_full_inst;
    output almost_full_inst;
    output full_inst;
    output error_inst;
    output [width-1 : 0] data_out_inst;

    // Instance of DW_fifo_s1_sf
    DW_fifo_s1_sf #(width, depth, ae_level, af_level, err_mode, rst_mode)
        U1 (.clk(inst_clk), .rst_n(inst_rst_n), .push_req_n(inst_push_req_n),
           .pop_req_n(inst_pop_req_n), .diag_n(inst_diag_n),
           .data_in(inst_data_in), .empty(empty_inst),
           .almost_empty(almost_empty_inst), .half_full(half_full_inst),
           .almost_full(almost_full_inst), .full(full_inst),
           .error(error_inst), .data_out(data_out_inst) );
endmodule

```

## Revision History

For notes about this release, see the [DesignWare Building Block IP Release Notes](#).

For lists of both known and fixed issues for this component, refer to the [STAR report](#).

For a version of this datasheet with visible change bars, click [here](#).

Date	Release	Updates
July 2020	DWBB_201912.5	<ul style="list-style-type: none"><li>Adjusted content and title of <a href="#">“Suppressing Warning Messages During Verilog Simulation”</a> on page 6 and added the DW_SUPPRESS_WARN macro</li></ul>
October 2019	DWBB_201903.5	<ul style="list-style-type: none"><li>Added the “Disabling Clock Monitor Messages” section</li></ul>
March 2019	DWBB_201903.0	<ul style="list-style-type: none"><li>Removed footnote about obsolete implementations from <a href="#">Table 1-3</a> on page 3</li></ul>
January 2019	DWBB_201806.5	<ul style="list-style-type: none"><li>Updated example in <a href="#">“HDL Usage Through Component Instantiation - VHDL”</a> on page 13</li><li>Added this Revision History table and the document links on this page</li></ul>



## Copyright Notice and Proprietary Information

© 2022 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

### Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

### Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

### Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <https://www.synopsys.com/company/legal/trademarks-brands.html>.

All other product or company names may be trademarks of their respective owners.

### Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

### Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.  
[www.synopsys.com](http://www.synopsys.com)

